Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primož Bevk

Odzivno spletno oblikovanje

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Damjan Vavpotič
Ljubljana 2013

Zahvaljujem se mentorju doc. dr. Damjanu Vavpotiču za pomoč in vodenje pri opravljanju diplomske naloge. Hvala tudi Michaelu Chipperfieldu iz grafičnega studia Magictorch in Rogerju Dudlerju iz podjetja Frontify za dovoljeno uporabo slik, ter družbi Telekom Slovenije za pomoč pri študiju. Posebna zahvala velja tudi staršem in ženi Metki, ki so me vseskozi spodbujali in mi omogočili študij.

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **PRIMOŽ BEVK**,

z vpisno številko **63080022**,

sem avtor/-ica diplomskega dela z naslovom:

Odzivno spletno oblikovanje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Damjana Vavpotiča

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja/-ice: _____

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00141/2013
Datum: 12.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PRIMOŽ BEVK**

Naslov: **ODZIVNO SPLETNO OBLIKOVANJE**
RESPONSIVE WEB DESIGN

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V okviru diplomskega dela predstavite tako imenovano odzivno spletno oblikovanje (responsive web design). Predstavite ključne tehnologije in pristope, ki se v tem kontekstu pojavljajo. Uporabo tehnologij in pristopov odzivnega spletnega oblikovanja predstavite na praktičnem primeru.

Mentor:

doc. dr. Damjan Vavpotič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Kazalo

Povzetek

S porastom uporabe mobilnih naprav, predvsem tabličnih računalnikov in pametnih telefonov, se hkrati tudi bliskovito povečuje število uporabnikov spleta na teh napravah. Za dobro uporabniško izkušnjo je bistvenega pomena, da je spletna stran dobro prilagojena velikosti zaslona naprave, s katero uporabnik dostopa do dotične spletne strani. Tudi navigacija po spletni strani in vsa interakcija morata delovati čim bolj naravno in intuitivno na vseh napravah, to pa je marsikdaj težko doseči. Še nedolgo nazaj je bila splošno uveljavljena rešitev tega problema postavitev dodatne spletne strani izključno za mobilne platforme, v zadnjem času pa postaja vedno bolj razširjen tako imenovan *Responsive Web Design*. Za razliko od starejših pristopov se odzivno spletno oblikovanje ne omeji zgolj na eno ločljivost zaslona in posledično na ozek spekter naprav, temveč je v svojem bistvu zastavljeno tako, da zaobjame vse ločljivosti in vse naprave.

V tem diplomskem delu bodo podrobneje predstavljene tehnologije, ki sestavljajo odzivno spletno oblikovanje, splošno uveljavljene pristope in smernice razvoja. Uporaba odzivnih pristopov bo podrobneje predstavljena na primeru prenove spletnega portala mladi.net, na katerem bodo spremembe in izboljšave tudi ovrednotene.

Ključne besede: Odzivno spletno oblikovanje, medijske poizvedbe, brskalnikovo okno, kontekstualno prilagajanje, progresivno izboljševanje, mobilnost na prvem mestu

Abstract

With an unprecedented increase in mobile phone and tablet usage, we are witnessing a rapid increase in internet users accessing the World Wide Web from these devices. Since a good user experience is paramount to ensure the visitor will enjoy the website, it is necessary to take steps to adapt the website to the various device screen sizes the users use. The same applies to the navigation and interaction on the said website. The flow of content and the whole user experience must feel natural and simple across all web capable devices, which is not particularly easy to achieve. Not so long ago, the go-to method was to create an additional website that would be served to mobile users, which has lately been replaced with a more dynamic approach dubbed Responsive Web Design. The main difference between the two is that the latter takes into account the whole spectrum of different screen resolutions and allows for the website to adjust its content and design accordingly and not just statically serve content specifically designed for a very narrow selection of devices.

The focus of the present thesis will be on technologies making up Responsive Web Design, different accepted methods and emerging development guidelines that will be presented in a case study of the ongoing redesign of the mladi.net website. The changes and improvements on the website will be also evaluated.

Keywords: Responsive Web Design, media queries, viewport, contextual adaptation, progressive enhancement, mobile first

1 UVOD

Od začetkov odzivnega spletnega oblikovanja, ki je bilo kot tako prvič poimenovano v članku Ethana Marcotta leta 2010 [31, 3], so minila že dobra tri leta. Avtor članka je dosegel skoraj popolno odzivnost spletne strani z združitvijo že obstoječih tehnologij in metod, s čimer se je začel trend spletnih strani, ki se prilagajajo odjemalcu oziroma uporabniku. Takrat se je že videlo, da bo uporaba mobilnih platform za dostop do interneta z vsakim letom vztrajneje naraščala in da bo izdelavo spletnih strani za ozek spekter različnih ločljivosti zaslona potrebno zamenjati s smotrnejšim razvojnim postopkom. Danes je odzivno spletno oblikovanje že skoraj norma pri načrtovanju in izdelavi nove sodobne spletne strani. Tudi večja podjetja, ki so pregovorno počasnejša pri odgovarjanju na nove trende in pristope, se vedno bolj zavedajo, da morajo ubrati pot odzivnega spletnega oblikovanja, če želijo obdržati in pridobiti nove uporabnike. Pričujoča naloga želi predstaviti spremembe in novosti na področju odzivnega spletnega oblikovanja, ki v krogih spletnih razvijalcev in uporabnikov pridobiva vse pomembnejšo vlogo. S krajšim pregledom tehnologij sodobnega spleta se bo dotaknila osnove, ki tvori odzivni splet, v naslednjih poglavjih pa jo bo vzela pod drobnogled in razčlenila ključne tehnike in pristope, ki so danes v uporabi. Za konec bodo povzete aktualne smernice razvoja, v zaključku pa bo podana študija primera, v kateri bo nekaj opisanih tehnik prikazanih na praktičnem primeru.

2 Tehnologije sodobnega spleta

V poglavju o tehnologijah sodobnega spleta bosta predstavljena standarda HTML5 in CSS3, ki sta trenutno na področju razvoja novih spletnih strani najbolj uporabljana. Oba prinašata veliko novosti in izboljšav, v nadaljevanju pa bodo podrobneje predstavljene le nekatere, ki so za odzivno spletno oblikovanje najpomembnejše.

2.1 HTML5

HTML5 v primerjavi s svojim predhodnikom HTML 4.01 vnaša v standard kopico sprememb [1], ki razširjajo uporabni vidik jezika HTML, hkrati pa naredi korak naprej z vidika uporabniške izkušnje z vpeljavo novih bolj opisnih semantičnih elementov in s poenostavljenim načinom vključevanja zunanjih virov.

Med novimi elementi lahko tako najdemo *header*, *article*, *footer*, *video* idr. – to so elementi s semantičnim pomenom, ki imajo po standardu W3C HTML5 definiran namen uporabe in imajo v novejših brskalnikih že privzeto definirane stile in akcije. Z njimi lahko nadomestimo splošno namenski element `<div>`, katerega smo v starejših standardih jezika HTML uporabljali za vpeljavo iste funkcionalnosti.

Primer uporabe novih semantičnih elementov:

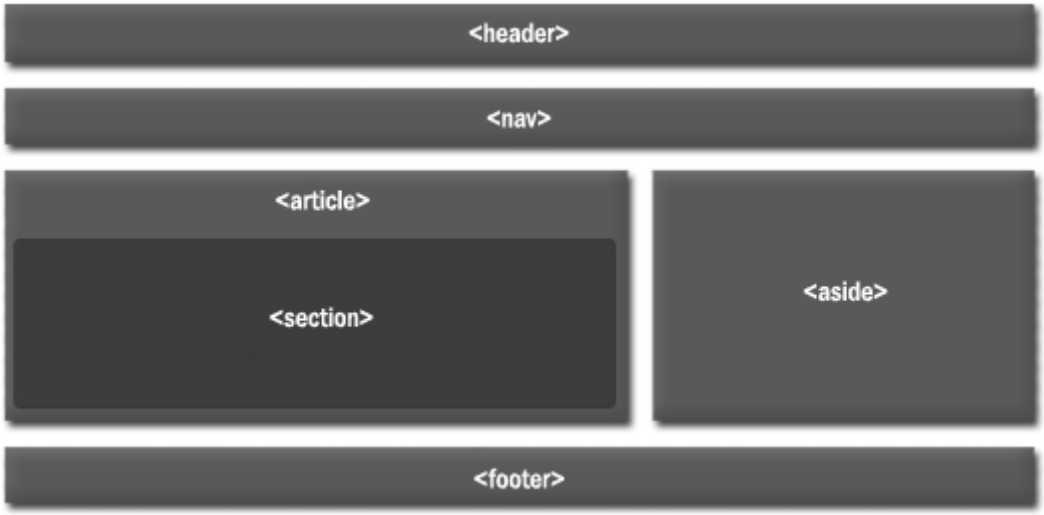
HTML 4.01

```
<div class="header">
      <div class="navigation">
            <ul class="menu">
                  <li><a href="#" title="Domov">Domov</a></li>
                  <li><a href="#" title="O meni">O meni</a></li>
            </ul>
      </div><!—konec navigacije -->
</div><!—konec glave -->
```

HTML5

```
<header>
      <nav>
            <ul id="menu">
                  <li><a href="#" title="Domov">Domov</a></li>
                  <li><a href="#" title="O meni">O meni</a></li>
            </ul>
      </nav>
</header>
```

Čeprav je na prvi pogled razlika zgolj v semantičnem zapisu (kljub bolj opisnemu imenu moramo vsakemu elementu še vedno definirati prekrivne sloge CSS), se za uporabniku razumljivimi imeni elementov skriva več prednosti. Prva takoj opazna prednost je v tem, da hitreje najdemo zaključni element določenega objekta (npr. `</header>`), kar olajša odkrivanje napak in pospeši postopek oblikovanja po predlogi, ki se uporablja pri sistemih za upravljanje z vsebino (*content management systems*). Pri uporabi elementa `<div>` smo v preteklosti pri kompleksnejših ugnezdenih zgradbah večkrat naleteli na mnogoštevilčno zaporedje `</div>` zaključnih elementov, zaradi česar je bilo ugotavljanje, kateremu delu strukture DOM pripada določen `</div>`, težavneje. Druga zelo pomembna prednost pa je, da opisni elementi ob pravilni uporabi sporočajo pomen posameznih delov dokumenta ne samo razvijalcu, ampak tudi brskalniku, in kar je morda celo pomembneje, tudi spletnim iskalnikom. Na podlagi opisnih lastnosti teh elementov lahko spletni iskalniki tako razčlenjene spletne strani bolje razumejo in posledično tudi ustrezneje razvrstijo v svojih iskalnih podatkovnih bazah.



Slika 1: Primer strukture strani HTML z uporabo opisnih elementov

Slika 1 prikazuje tipičen primer strukture enostavnejše spletne strani z uporabo opisnih elementov. Iz imen elementov na skici lahko brez težav ugotovimo, katera vsebina je kje in kako se med seboj povezuje.

HTML5 poleg že omenjenih pridobitev poenostavlja semantične zahteve pri dodajanju povezav na zunanje datoteke CSS in JavaScript. Atribut `type` tako ni več potreben, prav tako ni potrebno paziti na velike in male črke pri zapisu, po novem standardu je namreč veljavno oboje. Tudi deklaracija dokumenta ni več tako dolga, ampak je skrajšana na zapis `<!DOCTYPE html>`.

HTML 4.01

```
<script src="js/jquery-1.8.min.js" type="text/javascript"></script>
```

HTML5

```
<sCRipTsrC=js/jquery-1.8.min.js></scRiPT>
```

Vendar nova pravila niso zgolj modna muha in vesela novica za površne pisce kode, s poenostavitvijo zapisa lahko prihranimo kar nekaj odvečnih znakov, kar je pri razvijalcih bolj obiskanih strani, ki se poslužujejo različnih tehnik za optimizacijo datotek HTML, CSS in JavaScript, še kako dobrodošlo.

2.2 CSS3

Če smo z uporabo jezika HTML postavili strukturo strani, potem s prekrivnimi slogi ali CSS (*Cascading Style Sheets*) dodamo našim elementom HTML stilske lastnosti [2]. Z njim opisujemo lastnosti kot so barva, oblika, pisava, velikost in odnos med posameznimi elementi, ki skupaj tvorijo spletni dokument oziroma spletno stran.

CSS3 še ni v celoti potrjen, vendar je večina novosti, ki jih prinaša, že implementiranih v novejših spletnih brskalnikih, večinoma še v obliki eksperimentalnih implementacij, ki se od končne ločijo po specifičnih predponah za vsakega proizvajalca (npr. -webkit, -moz, -o, itd.). Z možnostmi, ki jih ponuja, se vedno bolj približuje željam in potrebam oblikovalcev spletnih strani, ki so za želene učinke v preteklosti morali večkrat poseči po kreativnih in kompleksnih rešitvah, ki so največkrat delovale samo v določenem spletnem brskalniku. Primer takega pomanjkanja podpore so na primer zaobljeni robovi. Tipičen postopek za dosego tega stilskega učinka z uporabo CSS2 je tako imenovana tehnika drsnih vrat (*sliding doors technique*), ki je zahtevala ugnezdeno strukturo HTML dveh elementov HTML. Z uporabo dveh slik, ki smo ju postavili eno na drugo, smo dosegli učinek elementa, ki se lahko prilagaja v eno od dveh smeri (širino ali višino) glede na vsebino in ima zaobljene robove. Element je bil lahko povsem preprost, na primer:

```
<style>

div {
      display: block;
      height: 30px;
      float: left;
      font-size: 1.2em;
      padding-right: 0.8em;
      background: url(img/bigbox.png) no-repeat scroll top right;
}
```

```
div span {
      display: block;
      line-height: 30px;
      padding-left: 0.8em;
      background: url(images/smallbox.png) no-repeat;
}
</style>

<div><span>Besedilo v škatlici</span></div>
```

»Škatlico« z zaobljenimi robovi smo ustvarili s pomočjo dveh slik. Elementu `<div>` smo za ozadje določili večjo sliko bigbox.png, ki je visoka 30 pikslov, široka 200 pikslov in ima na desni strani zaobljene robove. Elementu `<span>` pa smo za ozadje nastavili sliko smallbox.png s 30 piksli višine in 20 piksli širine. Vsaka od slik predstavlja polovico drsnih vrat, z dolžino 200 pikslov smo predvideli določeno mejo, do katere se vsebina elementa lahko razširi in pri kateri bo videz »škatlice« z zaobljenimi robovi še vedno dosežen. Nekaj prilagodljivosti torej imamo, vendar jo moramo že vnaprej predvideti, spreminjanje take »škatlice« pa zahteva precej dela, ker moramo za vsako spremembo barve ozadja ali radija zaobljenih robov ponovno ustvariti obe sliki. S CSS3 lahko ta pogost problem rešimo z uporabo lastnosti `border-radius`, s pomočjo katere se znebimo potrebe po uporabi slik, barvo ozadja pa preprosto določimo z lastnostjo `background-color`. Poleg enostavnejšega zapisa in hitrejšega časa nalaganja zaradi izpustitve dveh zahtevkov HTTP za slike, smo s to spremembo pridobili tudi na prilagodljivosti in ponovni uporabnosti kode.

3 Odzivno spletno oblikovanje

Slika 2: Odzivno spletno oblikovanje prilagaja vsebino uporabnikovi napravi [27]

Z vse zmogljivejšimi mobilnimi telefoni in pojavom cenovno ugodnih pametnih telefonov in tablic se število različnih dimenzij zaslona na trgu z vsako novo napravo vztrajno povečuje. Star pristop reševanja težave prikaza spletnih strani tako na osebnem računalniku kot na mobilni napravi je narekoval izdelavo dveh ločenih spletnih strani, osnovno za uporabnike osebnih računalnikov in okleščeno različico za uporabnike mobilnih telefonov. Še pred nekaj leti je bil tak pristop zadovoljiv, v zadnjem času pa postaja vedno bolj nevzdržen, ker je nesmiselno in neekonomično neprekinjeno prilagajati in izdelovati nove in nove iteracije iste spletne strani za vsako novo napravo, ki pride na trg. Kot odgovor na to se je pojavilo odzivno spletno oblikovanje (*Responsive Web Design*), ki združuje različne tehnike in dobre prakse izdelave spletnih strani, ki združene tvorijo odzivno spletno stran. Tako zgrajena spletna stran ni več namenjena ozkemu spektru naprav, ampak se dinamično in neodvisno prilagaja zaslonu oziroma mediju, na katerem je prikazana, ter podprtim funkcionalnostim uporabnikovega spletnega brskalnika kot voda, ki se oblikuje po predmetu, v katerega je nalita (Slika 2).

V nadaljevanju poglavja se bomo posvetili pomembnejšim tehnologijam in tehnikam, s pomočjo katerih je odzivno spletno oblikovanje sploh mogoče. Nekaj besed bomo namenili polyfilom (*Polyfills*), ki igrajo bistveno vlogo pri podpori in prikazu sodobne spletne strani na starejših, a še vedno zelo uporabljanih, spletnih brskalnikih, kot je na primer Internet Explorer 7. Osrednja tema bo CSS-ov modul medijskih poizvedb (*media queries*), zaradi katerega je odzivni splet pravzaprav sploh možen, ustavili pa se bomo tudi pri procesu oblikovanja spletne strani s pomočjo tipografskih mrež in priprave in umeščanja prilagodljivih slik v spletno stran.

3.1 Brskalnikovo okno

Odzivno spletno oblikovanje [3] se začne in konča pri brskalnikovem oknu (*viewport*), v katerem se vsebina našega spletnega dokumenta sestavi in izriše. Okno predstavlja meje prostora, v katerem lahko predstavimo vsebino, ki je uporabniku vidna. Jezik HTML pozna element `<meta>` z lastnostjo `viewport`, s pomočjo katere lahko brskalniku spletna stran sporoči, kako naj jo brskalnik v svojem oknu prikaže. Če ta element ni eksplicitno določen, spletni brskalnik poskrbi, da je spletna stran vedno v celoti vidna na zaslonu. To pomeni, da bo poskrbel, da je celotna širina spletne strani znotraj brskalnikovega okna, kar na mobilnih napravah pomeni, da bo le-ta krepko pomanjšana. Rezultat tega je skoraj nečitljivo besedilo vsebine, navigacija po taki strani pa zahteva dosti povečevanja in pomanjševanja ter premikanja po sami strani, kar je za uporabnika zelo zamudno in nepotrebno opravilo. Uporabnikova izkušnja na taki strani je slaba – stran, njeno vsebino in avtorja tako zelo hitro asociira z neprofesionalnostjo in postane nezaupljiv do nje. Negativnih čustev in asociacij pa si kot lastniki in razvijalci spletnih strani nikakor ne želimo pri svojih uporabnikih.

Prvi korak pri odzivnem spletnem oblikovanju je torej eksplicitno navodilo spletnemu brskalniku naj spletne strani, ki jih prejme od strežnika, ne prilagaja, ampak naj jih prikaže v pravi velikosti. Poleg tega lahko uporabniku tudi preprečimo povečevanje in pomanjševanje vsebine brskalnikovega okna ali pa ga omejimo z mejnimi vrednostmi. Slednje je pri odzivnem spletnem oblikovanju priporočljivo, ker bomo sami poskrbeli za prikaz vsebine v pravi obliki in velikosti pri vseh razmerjih zaslona, povečavo pa je z ozirom na uporabnike, ki imajo težave z vidom, smiselno obdržati, če menimo, da jim bo to koristilo.

3.2 Polyfili (Polyfills)

Slika 3: S pomočjo polyfilov lahko premostimo morje težav, ki jih za starejše brskalnike predstavljajo sodobne tehnologije [28]

Ker je nemogoče zagotoviti in pričakovati, da bo vsak uporabnik poskrbel za nenehno posodabljanje svojega računalniškega sistema in mobilne naprave, ter tako imel vedno najnovejšo različico svojega priljubljenega spletnega brskalnika, kot razvijalci spletnih strani naletimo na veliko oviro. Slika uporabnikov naše spletne strani ni enotna in v večini primerov ima več kot polovica uporabnikov zelo zastarelo programsko opremo. Kot dobri razvijalci želimo omogočiti vsem uporabnikom podobno dobro izkušnjo spletne strani, zaradi česar se večkrat raje odločimo za starejše tehnike in tehnologije, ki jih podpira širši krog spletnih brskalnikov. To pa nikakor ni rešitev, ki bi omogočala napredek.

Kot odgovor na to so se pojavili tako imenovani polyfili [4],ki so lahko v obliki aplikacije ali skripte, ki jo uporabnik prenese ali pa se skupaj s spletno stranjo prenese in premosti (Slika 3) funkcionalnosti, ki jih uporabnikov brskalnik ne podpira. S pomočjo knjižnic, kot je Modernizr [5], ki ga bomo podrobneje spoznali v poglavju o smernicah razvoja, lahko zaznamo, ali uporabnikov brskalnik podpira vse funkcionalnosti, ki smo jih implementirali v našo spletno stran. V primeru, da katere ne podpira, lahko naložimo ustrezen polyfil, s katerim

to funkcionalnost nadomestimo in uporabniku omogočimo polnejšo izkušnjo naše spletne strani.

Polyfili se najpogosteje uporabljajo v razvoju s tehnologijo HTML5, ker vnaša veliko novih tehnik in tehnologij, ki jih starejši brskalniki, predvsem Internet Explorer 8, 7 in 6, ne poznajo. Pri tem si lahko pomagamo tudi s spletno stranjo HTML5 Please [6], ki vzdržuje seznam slabo podprtih funkcionalnosti v različnih popularnih brskalnikih in navodila ter najprimernejše polyfile, s katerimi lahko manjkajoče funkcije nadomestimo.

3.3 Medijske poizvedbe CSS3 (CSS3 media queries)

Modul medijskih poizvedb CSS3 omogoča usmerjeno uporabo specifičnih stilov CSS glede na zmožnosti prikaza posamezne naprave. Z nekaj preprostimi definicijami lahko spremenimo videz vsebine glede na lastnosti zaslona naprave, kot so na primer širina, razmerje stranic ali orientacija zaslona.

Tip medija (*media types*) je v uporabi že od CSS2 dalje, pri katerem smo lahko razločevali stile glede na namembnost stiliziranega dokumenta, ki je lahko namenjen bodisi prikazu na zaslonu ali pa je oblikovan za tisk (*screen* in *print*). Z atributom `<link type="text/css" media="screen" href="stil.css">` smo lahko v glavi dokumenta določili, katera stilska predloga CSS naj se uporabi za oblikovanje vsebine v primeru, ko je ta prikazana na zaslonu (parameter *screen*), in katera v primeru, da je dokument poslan v tisk (parameter *print*). Ta lastnost je izhodišče za medijske poizvedbe v standardu CSS3, kjer se uporabnost razširja z dodatnimi testi lastnosti.

3.3.1 Sintaksa

V spodnjem primeru je prikaz uporabe medijske poizvedbe znotraj dokumenta CSS.

```
@media screen and (min-width: 320px) { /* CSS stili */ }
```

Sestavljen je iz uvodne označbe `@media`, ki ji sledi tip medija, za katerega naj poizvedba velja. Poizvedbo lahko naredimo poljubno specifično z dodajanjem testov za podrobnejše lastnosti naprave, kot na primer test ločljivosti zaslona. Posamezne teste lastnosti naprave dodajamo z veznikoma *in* in *ali* (*and* in *or*) tako, da oblikujemo logični pogoj, ob katerem želimo, da bodo stili, ki jih bomo v nadaljevanju dokumenta definirali, veljali. Znotraj zavitih oklepajev sledijo naše oblikovne lastnosti CSS, ki se nanašajo na elemente dokumenta HTML ob veljavnosti pogojev medijske poizvedbe.

Z ukazom `@import` lahko poizvedbo spremenimo tako, da stilsko predlogo raje uvozimo iz zunanjega dokumenta, kot da jo definiramo znotraj zavitih oklepajev v istem dokumentu. Pri obsežnih stilskih predlogah je to dobrodošla možnost, ki nam pomaga vzdrževati urejen in pregleden seznam stilov CSS, vendar se moramo zavedati, da vsaka vključitev predstavlja dodatno strežniško zahtevo, ki podaljša čas nalaganja našega dokumenta HTML.

```
@import url("stil320.css") screen and (min-width:320px);
```

3.3.2 Seznam testov

width	širina brskalnikovega okna (*viewport width*)
height	višina okna
device-width	širina zaslona naprave
device-height	višina zaslona naprave
orientation	orientacija zaslona S to lastnostjo lahko preverimo, ali je zaslon v pokončnem (*portrait*) ali ležečem (*landscape*) položaju.
aspect-ratio	Razmerje zaslona (okna), podano na podlagi širine in višine okna (*viewport*).
device-aspect-ratio	Podobno kot *aspect-ratio*, vendar za izračun uporablja širino in višino zaslona in ne okna.
color	število bitov na barvno komponento (16, 24 ali 32 bitov)
color-index	število vnosov v tabeli barv (*color look up*

	table) naprave
monochrome	število bitov na piksel v predpomnilniku monokromatskih strani (*monochrome frame buffer*)
resolution	ločljivost zaslona Sprejema vrednosti DPI (*dots per inch*) in DPC (*dots per centimetre*).
scan	Lahko je bodisi progresiven (*progressive*) ali prepleten (*interlaced*), z njim lahko naslovimo npr. televizijske zaslone, ki uporabljajo bodisi progresiven bodisi prepleten prikaz slike.
grid	Ločimo mrežne (*grid*) ali bitne (*bitmap*) zaslone.

Tabela 1: Seznam testov medijskih poizvedb, povzeto po [7]

Vse teste iz Tabela 1, z izjemo testov *grid* in *scan*, se uporablja z dodanima predponama *min* ali *max*, s pomočjo katerih lahko v pogojnih stavkih ustvarimo območja, za katera veljajo znotraj zavitih oklepajev definirane stilske lastnosti CSS.

3.3.3 Uporaba v odzivnem spletnem oblikovanju

Medijske poizvedbe so podprte v vseh najbolj razširjenih in uporabljanih spletnih brskalnikih (Firefox 3.6+, Safari 4+, Chrome 4+, Opera 9.5+, iOS Safari 3.2+, Opera Mobile 10+, Android 2.1+ in Internet Explorer 9+). Za starejše spletne brskalnike, predvsem za Internet Explorer 6, 7 in 8, pa obstajajo tako imenovani polyfili oziroma alternative, ki smo jih že opisali v prejšnjem poglavju. Testi, ki jih lahko dodajamo poizvedbam, so bistvo odzivnega spletnega oblikovanja. Zaradi svoje preprostosti in odzivnosti predstavljajo najbolj ekonomičen način, s katerim lahko pridemo do pomembnih informacij o napravi, s katero uporabnik dostopa do našega spletnega dokumenta in mu le-tega prikažemo v taki obliki, ki je zanj najbolj primerna.

Pri prikazu strani je za nas ključnega pomena podatek o velikosti posameznih elementov, ki sestavljajo spletno stran. Da bi lahko prikaz prilagodili različnim velikostim zaslona, moramo vzeti za primerjavo lastnost, ki nam da to informacijo, v našem primeru je to test širine zaslona. S pomočjo medijskih poizvedb lahko ustvarimo več skupin pravil CSS glede na razpon širine zaslona.

```
body {
     font-size: 26px;
}
@media screen and (max-width: 960px) {
     body {
          font-size: 24px;
    }
}
@media screen and (max-width: 768px) {
     body {
          font-size: 22px;
    }
}
@media screen and (max-width: 600px) {
     body {
          font-size: 18px;
    }
}
@media screen and (max-width: 480px) {
     body {
          font-size: 16px;
    }
}
```

S tem smo ustvarili skupine stilskih lastnosti, ki nam omogočajo določanje specifičnih prikazov za različne naprave. Tako bo na primer na napravi z ločljivostjo zaslona 320 × 480 (320 pikslov širine in 480 piksov višine) besedilo v telesu dokumenta HTML prikazano z velikostjo pisave 16 pikslov, kar ustreza pravilu:

```
@media screen and (max-width: 480px) {
     body {
          font-size: 16px;
    }
}
```

Hkrati se moramo zavedati, da bo zaradi načina delovanja CSS-a, preden bo velikost pisave nastavljena na 16 pikslov, brskalnik upošteval tudi vse medijske poizvedbe z veljavnimi pogoji, ki se nahajajo pred njo. Max-width: 600 px, max-width: 768 px itd. so prav tako veljavni za našo napravo s širino zaslona 320 pikslov, vendar so zaradi kaskadne lastnosti CSS-a prepisani z vsako naslednjo ponovno definicijo lastnosti, zaradi česar obvelja zadnja nastavitev velikosti pisave. V danem primeru je tak način zapisa popolnoma v redu, pri uporabi slik za ozadja elementov pa je že potreben razmislek kako najbolj smotrno definirati pogoje in stile, da ne bomo po nepotrebnem nalagali slik s strežnika. Medijske poizvedbe se tipično uporabljajo za proženje nalaganja manjših slik na naprave z manjšimi zasloni, pri katerih lahko z uporabo manjših in manj podrobnih slik prihranimo na času nalaganja, hkrati pa s tem ne poslabšamo uporabnikove izkušnje. Če bi v telo dokumenta (*body*) z zgornjim seznamom medijskih poizvedb CSS nalagali na primer sliko ozadja z uporabo lastnosti `background-image`, bi na naši napravi s širino zaslona 320 pikslov povzročili, da bi se v ozadju naložile slike iz vseh veljavnih pogojev, kar bi imelo precejšen vpliv na hitrost nalaganja strani in na količino prenesenih podatkov. Za reševanje tega problema se uporablja pristop mobilnosti na prvem mestu, ki je podrobneje opisan v poglavju o pristopih.

Opisan način združevanja stilov CSS v skupine za različne velikosti zaslonov naprav nam omogoča skoraj popoln nadzor nad obliko dokumenta. Tako lahko na primer na napravah z majhnim zaslonom poudarimo vsebino, zmanjšamo navigacijske elemente, odstranimo elemente, ki vsebujejo reklamna sporočila in elemente, ki vsebujejo informacije, ki niso toliko pomembne. Na napravah z večjimi zasloni, kot so na primer tablični računalniki in osebni računalniki, pa lahko dodajamo vedno več podrobnosti in elemente na strani drugače razporedimo, da dodatni prostor bolje izkoristimo. Gremo lahko celo tako daleč, da za manjše naprave pripravimo povsem drugo vsebino in grafično podobo, ki je skrita v elementih HTML, ki postanejo vidni šele pri določeni širini zaslona.

3.4 Tipografske mreže s prilagodljivim razmerjem (*fluid proportion based grids*)

Klasičen proces oblikovanja spletne strani uporablja tipografsko mrežo kot izhodišče za določitev razmerij med posameznimi vizualnimi gradniki spletne strani. Mreža ima fiksne dimenzije in razmerja, zaradi česar ima tudi grafična podoba, osnovana na njej, točno določene mere. Pred vzponom pametnih mobilnih telefonov in tablic ter pojavom hitrejših in cenejših mobilnih internetnih povezav so se oblikovalci spletnih strani odločali za izhodiščno velikost spletne strani na podlagi podatkov o najbolj razširjeni ločljivosti zaslona med uporabniki spleta [8]. Ta je bila zelo preprosto določljiva, ker so skoraj vsi obiskovalci spleta uporabljali osebne računalnike, ločljivost zaslonov pa je rasla skoraj vzporedno s padanjem cen na trgu večjih in zmogljivejših računalniških zaslonov. Danes ta slika ni več tako jasna. Število uporabnikov spleta, ki za svoje aktivnosti na spletu uporablja mobilne naprave vztrajno narašča [9]. Posledično tudi oblikovanje in razvoj spletnih strani nista več tako samoumevna. Razvoj več vzporednih spletnih strani za različne velikosti zaslonov ni smotrna, zato je moralo priti do sprememb pri osnovnem procesu oblikovanja.

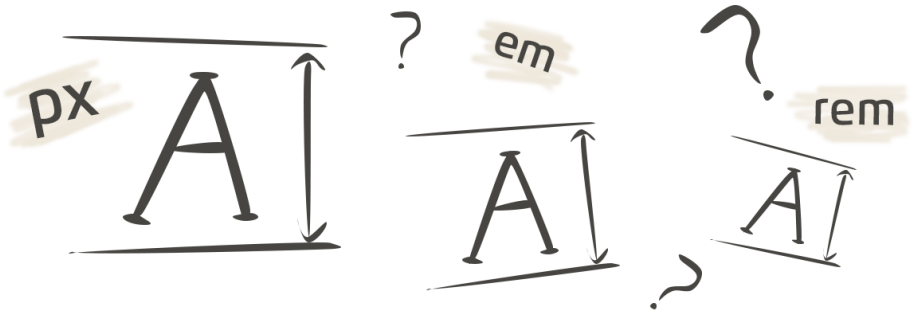
3.4.1 Tipografska mreža

Slika 4: Tipografska mreža s 16 stolpci

Kljub temu da že ves čas govorimo o odzivnosti in prilagodljivosti, se tipografske mreže s pojavom odzivnih spletnih strani niso prav nič spremenile. Še vedno so fiksne širine z enakomernimi odmiki in velikostmi stolpcev. Danes je najpogosteje uporabljana širina 960

pikslov, za specifične oblikovne potrebe pa je širina lahko seveda tudi drugačna. Običajno je mreža razdeljena na 12, 16 ali 24 stolpcev z dodanimi odmiki, kot naš primer na Slika 4, elementi pa so največkrat večkratniki širine osnovnega gradnika mreže, s čimer dosežemo uravnoteženost grafične podobe.

3.4.2 Kontekstualno prilagajanje



Slika 5: Velikost pisave lahko določimo v pikslih, tiskarskih emih, ali v relativnih remih [29]

Ker razvijamo spletno stran, ki bo dostopna uporabnikom najrazličnejših naprav in spletnih brskalnikov, moramo najprej poskrbeti za skupno izhodišče naše spletne strani. Vsak spletni brskalnik ima namreč lastne privzete nastavitve za odmike posameznih elementov HTML in svojo osnovno velikost pisave, ki je pri večini spletnih brskalnikov 16 pikslov. Zaradi tega se je uveljavila tako imenovana ponastavitev CSS (*CSS reset*) [10], ki nastavi skupno izhodišče vsem elementom in enotno velikost pisave, zaradi česar lahko z lastnimi pravili CSS dosežemo zelo podobno obnašanje na vseh napravah na različnih kombinacijah operacijskih sistemov in spletnih brskalnikov. Od tukaj naprej vsa razmerja in velikosti določamo kontekstualno oziroma v odvisnosti od starša elementa oziroma globalnih nastavitev, ki smo jih pravkar ponastavili.

Ker želimo, da je naša spletna stran odzivna, za mere ne uporabljamo več pikslov, ampak odstotke in eme. Em je tipografska enota, ki določa velikost pisave glede na velikost starša. Pri ponastavljenem CSS-u, kjer je `font-size: 100%`, kar v večini spletnih brskalnikov pomeni širino 16 pikslov, je 1 em enak 16 pikslom, 2 ema sta enaka 32 pikslom in 0,5 ema je enako 8 pikslom. Tako bomo na primer za naslove `h1` uporabljali velikost pisave 2 ema (32

pikslov), za podnaslove `h2` velikost 1,7 ema (27 pikslov) in za odstavke `p` velikost 1 em (16 pikslov). Če želimo v naslovu `h1` imeti del naslova v velikosti 24 pikslov, bomo ta del zavili v element `<span>` in določili stil `h1 span` z lastnostjo `font-size: 0.75em`.

Izračun: `16px * 2em = 32px 32px * 0.75em = 24px`

Če relativne velikosti pisave glede na starša ne potrebujemo in nam povzroča nepotrebne preglavice pri določanju vrednosti, lahko uporabimo tipografsko enoto rem. Rem je tako kot em relativna enota za določanje velikosti pisave s to razliko, da kot osnovo ne jemlje velikost starša, ampak osnovno velikost pisave telesa dokumenta HTML.

Tudi pri določanju velikosti posameznih elementov spletne strani, kot so na primer menijska vrstica, gradnik z vsebino, okvir z galerijo in podobno, bomo vedno izhajali iz našega osnovnega okvirja strani in velikosti vseh ostalih elementov določali relativno glede na njihovega starša. Če smo za osnovo vzeli tipografsko mrežo širine 960 pikslov, je najverjetneje tudi naš osnovni okvir strani enake širine, zato ga bomo definirali tako:

```
#telo {
     margin: 0 auto;
     width: 960px;
}
```

V nadaljevanju želimo v naše `#telo` postaviti blok z vsebino, ki je v našem dizajnu širok 680 pikslov, na njegovi levi strani pa želimo imeti tudi blok, v katerem se bo nahajala naša navigacija, ta naj bo širok 260 pikslov.

```
#navigacija {
     width: 260px
     float: left;
}
#vsebina {
     width: 680px
     float: right;
}
```

Tak zapis je pravilen, vendar ni odziven. Ker želimo, da ostaja razmerje velikosti med obema gradnikoma enako, hkrati pa želimo, da se prilagaja velikosti njunega starša (`#telo`), moramo njuni širini izraziti v odstotkih. Za izračun uporabimo formulo:

element / kontekst = rezultat

V našem primeru je element širina našega elementa oziroma gradnika, kontekst je njegov starš, rezultat pa je širina v razmerju do starša, izražena v odstotkih.

```
#navigacija {
     width: 27.0833333333; /* 260/960 */
     float: left;
}
#vsebina {
     width: 70.8333333333; /* 680/960 */
     float: right;
}
```

Pri uporabi formule za izračun kontekstualne širine pogosto dobimo zapis z veliko decimalnimi mesti. Zaokroževanje v takih primerih ni smiselno – širino je bolje podati s čim večjo natančnostjo, ker lahko zaokroževanje v določenih primerih predstavlja v končnem izdelku kar veliko odstopanje od želenega učinka.

Popraviti moramo tudi naš osnovni gradnik, `#telo`, da bo odziven in se bo prilagajal velikosti zaslona. V dizajnu smo določili, da je naša idealna širina gradnika `#telo` 960 pikslov, zdaj pa moramo še določiti, kaj se zgodi na zaslonih, ki v širino merijo manj.

```
#telo {
     margin: 0 auto;
     max-width: 960px;
     width: 90%;
}
```

Določili smo, da želimo, da je širina gradnika `#telo` enaka 90 % njegovega starša, elementa `body`, ki se razteza čez celotno širino zaslona. To razmerje je bilo določeno po občutku, kar je vizualno za naš testni dizajn najbolj primerno, in je seveda lahko različno od dizajna do dizajna. Hkrati smo dodali še eno pomembno lastnost, `max-width`, ki določa, da širina gradnika `#telo` ne sme nikoli preseči 960 pikslov, tudi če to pomeni, da ne bo več zadostila pravilu `width: 90%`, kar se bo zgodilo pri zaslonih kjer 90 % širine elementa `body` predstavlja večjo vrednost od 960 pikslov.

3.5 Prilagodljive slike

Slika 6: Primer prilagodljive slike na odzivni spletni strani [30]

Ljudje smo vizualna bitja, zato je grafična vsebina zelo pomembna za bralno razumevanje in hitro dojemanje pomembnih informacij, ki jih želimo posredovati obiskovalcu naše spletne strani. Fotografije, slike in drugi grafični elementi vedno bolj pridobivajo na pomembnosti v strukturi in dizajnu spletnih strani. To se izraža v trendu, ki je v zadnjih dveh letih privedel do preobrata, ki je v ospredje postavil vizualno vsebino pred tekstovno. Vedno več spletnih strani danes vsebino prilagaja slikovnim elementom, in ne več slik besedilu oziroma tekstovni vsebini. Tak pristop pa v sodobnem času zahteva odzivnejše prilagajanje slik kontekstu, v

katerega so umeščene, in posledično tudi ločljivostim zaslonov, na katerih so prikazane, kot je ponazorjeno na Slika 6.

Poglejmo si element `<img>`, s katerim v telo spletne strani vstavimo sliko:

```
<img src="img/primer1.jpg" alt="Primer 1" />
```

Po navadi takemu elementu definiramo še parametra za višino (*height*) in širino (*width*), pri odzivnem spletnem oblikovanju pa namesto tega določimo naslednjo lastnost CSS:

```
img {
     max-width: 100%;
}
```

Lastnost `max-width: 100%` pove spletnemu brskalniku, da lahko vse slike raztegne po celotni širini kontekstualnega elementa, v katerem se nahajajo. V primeru, da je kontekst slike telo spletne strani (*body*), bo širina slike vedno enaka širini brskalnikovega okna in se bo s povečevanjem ali manjšanjem okna dinamično povečevala oziroma pomanjševala, tako da bo vedno zapolnila celotno širino. To pa ni nujno naš željen končni rezultat. Slike imajo točno določeno velikost, in če sliko povečamo prek njene originalne velikosti, začne slika izgubljati ostrino. Temu se lahko izognemo tako, da naša pravila CSS za sliko dopolnimo. Ker je velikost posamezne slike specifična, bomo uvedli za vsako tako vstavljeno sliko svoj id CSS.

```
<img id="primer1" src="img/primer1.jpg" alt="Primer 1" />

<style>
#primer1 {
     width: 100%;
     max-width: 1280px;
}
</style>
```

Na tak način smo določili sliki primer1.jpg, da vedno zasede 100 % širine svojega starša, vendar samo do širine 1280 pikslov, ki v primeru slike primer1.jpg predstavlja originalno širino slike. S takim pristopom zagotovimo, da bo slika vedno primernih dimenzij glede na velikost zaslona naprave, ki jo uporabljamo, vendar s tem še nismo dosegli zadovoljive implementacije prilagodljive slike. Še vedno namreč uporabljamo eno velikost slike za vse velikosti zaslonov, kar je zelo potratno in pri manjših napravah, predvsem pri mobilnih telefonih z manjšim zaslonom in počasno internetno povezavo prek mobilnega omrežja. Za mobilne telefone bi bilo bolj smiselno imeti to isto sliko v velikosti 320 × 480 pikslov, kot pa 1280 × 1920 pikslov. Prav tako bi bilo smotrno imeti na razpolago še različico slike v velikosti 640 × 960 pikslov za mobilne telefone z večjo ločljivostjo zaslona. S temi tremi izrezi iste fotografije lahko napravi, ki dostopa do naše spletne strani, serviramo sliko, ki je glede na dimenzije zaslona naprave zanjo najbolj primerna. S prej definiranim pravilom CSS pa zagotovimo, da bo slika vedno zavzela celotno širino svojega starša. Ker bo velikost servirane slike optimalna za vsako napravo, bo zaradi pravila `max-width: 100%` raztezanje oziroma krčenje minimalno in ne bo bistveno vplivalo na videz slike. Bistvena pridobitev, poleg odzivnega prikaza, pa bo hitrejši čas nalaganja za uporabnika, ki se bo poznal predvsem na mobilnih telefonih z internetnim dostopom prek mobilnega omrežja, za katerega so značilne počasne povezave. Taka naprava bo s strežnika prenesla manjšo sliko, zaradi česar se bo celotna stran hitreje naložila, ob koncu dneva pa bo tudi za naš strežnik to pomenilo bistven prihranek v porabi pasovne širine.

Da strežnik pripravimo do serviranja različnih velikosti slik v pravih situacijah, se moramo poslužiti priročnega postopka in skripte za adaptivne slike, Adaptive Images [11]. S pomočjo sprememb v Apachejevi datoteki .htaccess poskrbimo, da vsak element `<img>`, ki ga brskalnik prebere, povzroči nalaganje datoteke PHP z algoritmom, ki glede na nastavljene parametre in velikost uporabnikovega zaslona ustvari iz originalne slike različico primernih dimenzij. Skripta to na novo ustvarjeno sliko shrani, da jo lahko strežnik ob naslednji poizvedbi servira in tako skrajša in optimizira proces.

4 Pristopi

4.1 Postopna degradacija in progresivno izboljševanje

Postopna degradacija (*graceful degradation*) [12], največkrat poimenovana tudi kot toleranca napak (*fault-tolerance*), je lastnost, ki omogoča sistemu, da nemoteno deluje kljub morebitnim napakam. Degradacija takega sistema je po navadi sorazmerna z velikostjo in obsegom napake. Jezik HTML je zasnovan na pristopu postopne degradacije, ker gre v osnovi za jezik, ki je naprej združljiv (*forward compatible*), kar pomeni, da spletna stran, narejena za najnovejši brskalnik, prav tako deluje v starejših. Nove elemente in lastnosti, ki jih ne razume, starejši brskalnik preprosto ignorira, zaradi česar je spletna stran še vedno dostopna, vendar zelo okrnjena. Spletne strani, ki so razvite po principu postopne degradacije, se praviloma razvijajo za najnovejše brskalnike. V zadnji fazi razvoja se take spletne strani testira tudi na starejših različicah brskalnikov, ki so še vedno v uporabi. Ker je nemogoče doseči enak videz in nabor funkcionalnosti spletne strani, se večinoma poskrbi za popravilo večjih napak in prilagodi videz, da pomanjkljivosti v prikazu niso tako izrazite.

Progresivno izboljševanje (*progressive enhancement*) [13] je koncept v katerem postavimo vsebino na prvo mesto. Zavedati se moramo, da je množica uporabnikov spleta zelo raznolika in da jih ne smemo gledati zgolj skozi ozek filter lastnosti brskalnikov, ki jih uporabljajo. Za brskalniki se skrivajo pravi ljudje z zelo različnimi merili za estetiko, ki niso prišli na našo spletno stran zaradi njenega videza (čeprav je tudi ta zelo pomemben), temveč zaradi njene vsebine. Poleg njih pa ne smemo pozabiti tudi na iskalne robote, ki pregledujejo vsebino za vključitev v indeks spletnih iskalnikov in za katere je pomembnejša dostopnost in razumljivost vsebine kot pa vizualni efekti.

S tem pristopom ločimo spletno stran na več smiselnih slojev. Vsebina predstavlja jedro in osnovo na kateri gradimo, to je semantično bogat dokument HTML. Naslednji sloj zajema vizualizacijo, kar je pri spletnih straneh večinoma CSS. Zadnji sloj pa JavaScript, s katerim poskrbimo za boljšo uporabniško izkušnjo. Če vse skupaj združimo, dobimo bogato uporabniško izkušnjo spletne strani, z odstranjevanjem posameznih slojev pa ne izgubimo na sporočilnosti, ker v svoji okrnjeni različici jedro še vedno ohrani svojo semantično bogato vsebino.

4.2 Mobilnost na prvem mestu (*mobile first*)

Odzivno spletno oblikovanje zaradi svoje narave, ki temelji na prilagodljivosti, omogoča več pristopov, s katerimi se lahko lotimo razvoja odzivnih spletnih strani. V ospredju je mobilnost na prvem mestu, ki preliva v prakso v prejšnjem poglavju omenjeno progresivno izboljševanje in ga razširi s progresivnim dodajanjem gradnikov in vizualnih elementov vzporedno s povečevanjem velikosti zaslona naprav [14, 25].

Medijske poizvedbe imajo ključno vlogo. Z uporabo pogoja `min-width` s postopnim povečevanjem širine zaslona dosežemo, da se na napravah z manjšim zaslonom, ki so praviloma počasnejše, naloži najmanj vizualnih elementov, ti pa se progresivno dodajajo na večjih zaslonih. S tem tudi dosežemo, da se na starejših različicah Internet Explorerja servira oblika spletne strani za naprave z majhnimi zasloni, ki v večini primerov uporablja preprostejše gradnike in vizualne efekte, ki so jih tudi starejši brskalniki zmožni prikazati brez večjih težav. Če to ni želen učinek, lahko še vedno uporabimo polyfile, ki omogočajo medijske poizvedbe tudi v starejših različicah Microsoftovega brskalnika. Z njihovo pomočjo poskrbimo, da se prek lastnosti CSS `background-image` oziroma lastnosti `background` skladno z velikostjo zaslona naložijo slike z različno ločljivostjo, ali pa na določenih ločljivostih zaslona popolnoma zamenjamo grafično podobo strani. Čeprav zveni enako kot postopek adaptivnih slik iz poglavja o prilagodljivih slikah, se slike, dodane prek lastnosti `background`, obnašajo drugače. Kot je iz imena lastnosti razvidno, imamo opravka z ozadji elementa, in ne s sliko kot elementom. Elementu z ozadjem lahko dodamo otroka z novo vsebino, ki deloma ali v celoti prekrije ozadje. Slika v ozadju poleg tega ne vsebuje opisnih atributov, kot so naslov, opis in alternativni tekst. Brez opisnih atributov take slike nimajo vrednosti za uporabnika, ki si ogleduje stran v tekstovnem brskalniku. Prav tako so take slike nevidne v primeru slabo definirane poti do slike, kjer se pri elementu `<img>` namesto slike prikaže alternativni tekst. In kar je morda še bolj pomembno, spletni iskalniki takih slik ne zaznajo oziroma jih ignorirajo, ker jih ne dojemajo kot del vsebine spletne strani. Tak način vključevanja slik je neprimeren za slike, ki so del vsebine, je pa nepogrešljiv za realizacijo vizualizacije vsebine oziroma za njeno predstavitev. S pristopom, ki postavlja mobilnost na prvo mesto, poskrbimo, da se bo vedno najprej naložila najmanjša ločljivost posamezne slike, pri napravah z večjimi zasloni pa se bodo izvedle tudi sledeče medijske poizvedbe, ki še imajo veljavne pogoje `min-width`, s pomočjo katerih se bodo naložile večje različice slik.

```
@media screen and (min-width: 320px) {
    #slika1 {
          background-image: url(img/slika1-320.jpg);
    }
}
@media screen and (min-width: 480px) {
    #slika1 {
          background-image: url(img/slika1-480.jpg);
    }
}
@media screen and (min-width: 767px) {
    #slika1 {
          background-image: url(img/slika1-767.jpg);
    }
}
@media screen and (min-width: 960px) {
    #slika1 {
          background-image: url(img/slika1-960.jpg);
    }
}
```

Zaradi narave CSS-a se bodo pri zaslonu z ločljivostjo 1280 × 1920 pikslov v element `#slika1` po vrsti naložile slike pri vseh zgoraj definiranih pogojih. To pomeni daljši čas nalaganja in odvečne strežniške zahtevke pri velikih zaslonih in manjši čas nalaganja in manj zahtevkov pri manjših zaslonih, kar je zelo dober kompromis. Večji zasloni so po navadi na zmogljivejših napravah, kot so osebni računalniki, prenosniki in tablični računalniki, ki so po navadi priklopljeni na hitrejše internetne povezave. Zaradi tega dodatni prenosi, ki se zgodijo v ozadju, ne vplivajo toliko na čas nalaganja spletne strani. Manjše zaslone pa najdemo na mobilnih telefonih, ki so večinoma vezani na počasnejše mobilne internetne povezave, zaradi česar je nujno, da prenesemo čim manj podatkov s čim manjšimi velikostmi, da lahko zagotovimo dovolj veliko odzivnost spletne strani.

5 Smernice razvoja

Če je bil še pred desetletjem svetovni splet zgrajen iz enostavnih spletnih strani, ki so uporabljale peščico dopolnjujočih se tehnologij, je danes slika precej drugačna. Statične strani so zamenjale kompleksne aplikacije, ki z naprednimi uporabniškimi vmesniki in zmožnostmi konkurirajo klasičnim programom. Z razmahom računalništva v oblaku in poceni servisov za hrambo podatkov smo vedno bliže točki, ko klasičnih programov ne bomo več uporabljali in bomo vse, kar potrebujemo, imeli na spletu, vedno dostopno kjer koli, kadar koli in iz katere koli naprave. Odzivno spletno oblikovanje gre z roko v roki s to vizijo modernega spleta in igra pomembno vlogo pri omogočanju dostopa do teh servisov čim širšemu krogu naprav in uporabnikov. Skupaj s to vizijo se razvijajo nove tehnologije in pristopi ter tehnike optimizacije, s pomočjo katerih se povečuje dostopnost in uporabnost spletnih strani ter hitrost njihovega razvoja.

5.1 Knjižnice in zbirke orodij

Spletne strani lahko razvijamo na več načinov. Lahko se poslužimo urejevalnikov WYSIWYG (*what you see is what you get*) ,v katere postavimo gradnike in nam sami ustvarijo kodo, ki je po večini neberljiva in polna nepotrebnih elementov in znakov, lahko pa jo sami spišemo. Ker je to precej dolgotrajen postopek, a kljub vsemu še vedno najbolj zanesljiv za optimalno spisano spletno stran, so se pojavile različne knjižnice in orodja, s katerimi lahko razvijalci enostavneje in hitreje postavijo in optimizirajo spletno stran. V nadaljevanju bo predstavljenih nekaj trenutno najbolj uporabljanih in uporabnih orodij, ki se uporabljajo v odzivnem spletnem oblikovanju.

5.1.1 LESSCSS

CSS je kaskadni stilski jezik, ki se izvaja v linearnem zaporedju in ne pozna funkcij in spremenljivk. Zaradi teh lastnosti se v razvoju večkrat zgodi, da moramo kakšen skupek lastnosti, ki jih želimo dodati več elementom, večkrat ponoviti, ker jih ne moremo zapisati v obliki generične funkcije, ki bi jo lahko uporabili v več primerih. Podobno težavo imamo pri vrednostih. Isto barvo, ki jo uporabljamo skozi celoten dizajn, moramo vsakič znova zapisati v svojem heksadecimalnem zapisu, namesto da bi preprosto uporabili spremenljivko. Ni potrebno veliko premišljevati, da se zavemo, kakšna težava nastopi, če želimo zamenjati eno

barvo z drugo. Namesto spremembe ene spremenljivke je potrebno iskanje po celotni datoteki in sprotno preverjanje novih napak, ki jih na tak način še prehitro vpeljemo. Kot ena izmed rešitev na tem področju je LESSCSS [15], ki v CSS vpelje spremenljivke, mixine, operacije in funkcije. Okorni, statični CSS kar naenkrat postane dinamičen, preprost in izredno hiter za razvoj. S pomočjo tako imenovanih mixinov lahko pripravimo skupke lastnosti, ki jih lahko večkrat uporabimo, z uporabo funkcij pripravimo dinamične stilske predloge, ki jih lahko preprosto in hitro prilagajamo, s čimer pomembno prihranimo na času. Z deklaracijo nekaj ključnih globalnih spremenljivk lahko vnaprej definiramo pomembne barve, odmike in druge vrednosti, ki jih lahko uporabljamo povsod, brez nenehnega razmišljanja o vrednostih, ki smo jih že definirali, in pogostega pregledovanja kode, če smo na kakšnem mestu zapisali napačno vrednost kakšnega odmika ali pisave.

5.1.2 Bootstrap

Bootstrap [16] je Twitterjevo ogrodje za hitro prototipiranje in razvoj odzivnih spletnih strani, ki postavljajo mobilnost na prvo mesto. Gre za nabor stilov CSS in funkcij JavaScript, s katerimi lahko hitro pripravimo osnovno tipografsko mrežo in vanjo postavimo že grobo oblikovane osnovne elemente, kot so gumbi, menijske vrstice, seznami, galerije, modalna pogovorna okna, okna z obvestili, obrazci itd. Zelo pomembna lastnost je, da so vsi ti elementi ob vključitvi odzivne knjižnice, ki je del Bootstrapovega paketa, odzivni in prilagodljivi na velikost zaslona uporabnikove naprave. Vse, kar nam še ostane, je, da poskrbimo za vsebino in dodamo svojo lastno stilsko predlogo in implementiramo morebitne dodatne funkcionalnosti.

5.1.3 Modernizr

Ob vedno večjem naboru naprav z dostopom do interneta in vizualno ter funkcionalno vedno bogatejšimi spletnimi stranmi se vedno pogosteje srečujemo z vprašanjem, komu lahko kaj prikažemo, ne da bi na ta način poslabšali uporabniško izkušnjo. Modernizr [17] se zažene ob začetku nalaganja spletne strani in izvede test uporabnikovega brskalnika za podporo različnih funkcionalnosti. Rezultate shrani v objekt JavaScript in kot razrede CSS, ki so dodani elementu `<html>`. Tako lahko v nadaljevanju s preprostim dostopom do teh lastnosti, bodisi

z uporabo pogojnih stavkov JavaScript ali pa s stili CSS, poskrbimo, da vsakemu uporabniku omogočimo zgolj tisto, za kar vemo, da bo njegov brskalnik lahko prikazal.

5.2 Optimizacija

Pred objavo spletne strani na spletu se moramo posvetiti še optimizaciji. S performančnega vidika se optimizacija večinoma vrši na podatkovnih bazah in mehanizmih za serviranje vsebine. Uporabniški vmesnik oziroma *front-end* pri grafično in funkcionalno bolj zahtevnih spletnih straneh pa prav tako ponuja veliko prostora za optimizacijo. Glavna cilja pri optimizaciji sta zmanjšanje velikosti datotek, ki jih serviramo in zmanjšanje števila zahtevkov HTTP, ki so potrebni za prikaz strani. Spodaj opisane metode pa se ne uporabljajo samo v odzivnem spletnem oblikovanju, ampak v spletnem razvoju na splošno in veljajo zaradi svoje uveljavljenosti za splošna vodila pri razvoju spletnih strani v današnjem času.

5.2.1 Stiskanje slik

Vizualizacija vsebine je za dojemanje in razumevanje sporočila ključna. Pri tem igrajo slike zelo pomembno vlogo, te pa imajo običajno kar zajetno velikost, na spletni strani pa po navadi nimamo samo ene, ampak več. To nas hitro pripelje do ugotovitve, da moramo nekaj narediti, sicer bo uporabnik moral predolgo časa čakati, da se mu bo naša spletna stran naložila, mi pa bomo imeli na koncu meseca zaradi tega velik račun za porabljeno pasovno širino. V prejšnjih poglavjih smo se že srečali s tehnikami, s pomočjo katerih lahko poskrbimo, da počasnejšim in manjšim napravam serviramo manjše slike, večjim pa večje, s čimer del tega problema rešimo. Te iste slike pa lahko še dodatno optimiziramo, in tako še dodatno prihranimo na prenosu podatkov.

Na spletu si ne moremo privoščiti serviranja slik v njihovi najvišji kvaliteti, zato poznamo v programih za urejanje slik, kot je na primer Photoshop, možnost shranjevanja slik za splet (*save for Web*). S pomočjo tega orodja lahko slike JPG shranimo v znižani kvaliteti, običajno v 40-odstotni oziroma od primera do primera drugače, s čimer poiščemo najboljše razmerje med končno velikostjo slike in njeno kvaliteto. Pri slikah z omogočeno prosojnostjo pa je to opravilo težje. Sem sodijo datoteke GIF in PNG, ki so najbolj v uporabi. Datoteke GIF imajo

omejeno paleto barv na 256 odtenkov, zaradi česar imajo od vseh naštetih formatov najmanjšo velikost, zato jih praviloma ni potrebno še dodatno optimizirati. Datoteke PNG moramo največkrat shranjevati v 24-bitni različici, ker se v 8-bitnem načinu izgubijo mehki robovi in sence postanejo ostre. Ker 24-bitna datoteka PNG vsebuje približno 3-krat več podatkov kot 8-bitna, je velikost datoteke seveda bistveno večja kljub Photoshopovemu algoritmu za stiskanje. To lahko rešimo z orodji, kot je TinyPNG [18], spletni servis, ki omogoča hitro, v povprečju za 60 % stisnjeno datoteko PNG ob ohranitvi približno enake kvalitete slike, vključno s prosojnostjo.

Stiskanje slik seveda ni vedno možno, sploh kadar moramo zagotoviti največjo kvaliteto slike z dovolj veliko ostrino ali pa poskrbeti, da ohranimo vse sence in mehko prosojnost. Take zahteve pa so po navadi večinoma na straneh, ki strežejo slike in druge grafične elemente, ki so namenjeni tisku, kjer je najvišja kvaliteta nujna za dovolj dober končni izdelek, za vse ostale namene se lahko uporabi stisnjene različice slik.

5.2.2 Zlepki slik (CSS sprites)

Poleg stiskanja slik, s katerim zmanjšamo njihovo velikost, lahko pri slikah, ki so večkrat uporabljene v našem dizajnu, naredimo še korak več in zmanjšamo tudi število strežniških zahtevkov. To dosežemo z uporabo zlepka slik (*CSS sprite*) [19], pri čemer v eno sliko združimo več različnih slik, zaradi česar v enem prenosu prenesemo več slik, ki jih iz zlepka posamično izločimo z nekaj pravili CSS. Ta tehnika se po navadi uporablja za navigacijske elemente, kot so gumbi, ikone in drugi deli, ki sestavljajo dizajn naše spletne strani, redkeje pa ga zasledimo pri slikah, ki sestavljajo vsebino.

Vzemimo za primer zlepek navigacijskih ikon.

Slika 7: Primer zlepka

Na Slika 7 so združene ikone po vrsti od leve proti desni. V našem dokumentu HTML imamo definirana dva gumba za navigacijo, levo in desno.

```
<a href='#levo' class='navigacija' id='levo'>levo</a>
<a href='#desno' class='navigacija' id='desno'>desno</a>
```

Naša datoteka CSS pa vsebuje naslednje lastnosti, s pomočjo katerih vsakemu elementu `<a>` dodamo svojo sliko iz zlepka nav.png.

```
.navigacija {
     background: url('img/nav.png');
     background-repeat: no-repeat;
     width: 40px;
     height: 40px;
     display: inline-block;
}
#levo {
     background-position: 0 0;
}
#desno {
     background-position: -40px 0;
}
```

V isti zlepek bi lahko vključili še več slik in jih na enak način določili tudi drugim elementom HTML.

Kako pa zlepke naredimo? Operacija je zelo preprosta. V svojem najljubšem urejevalniku slik odpremo vse slike, ki jih želimo združiti, vsako posebej izrežemo in prilepimo v eno veliko sliko. Zapomnimo si koordinate levih zgornjih kotov, na katerih se posamezne slike nahajajo znotraj našega zlepka, in jih vpišemo kot lastnosti `background-position` elementov HTML, katerim želimo te slike dodati kot ozadje. Proces lahko poenostavimo z uporabo spletnih servisov, ki omenjeni zlepek sami ustvarijo iz podanih slik, nekateri boljši, kot na primer SpriteMe [20], pa nam poleg tega tvorijo tudi pripadajoče lastnosti CSS, s katerimi dostopamo do slik v zlepku.

5.2.3 Minimizacija z uporabo orodja UglifyJS

Datoteke HTML, CSS in JavaScript, iz katerih je sestavljena večina spletnih strani, so v primerjavi z velikostjo slik, zvočnih datotek in videov razmeroma majhne, a jih lahko vseeno še zmanjšamo. Zakaj pa bi to pravzaprav počeli, če lahko z optimizacijo slik pridobimo neprimerno več? Vzemimo za primer najbolj obiskano spletno stran v Sloveniji, 24ur.com, ki je imela v juliju 2013 zavidljivih 91.491.399 prikazov strani samo v Sloveniji [21]. Ob nalaganju spletne strani brskalnik prenese za 116 KB datotek HTML, 146 KB datotek CSS in kar 473 KB datotek Java Script, kar smo zabeležili z uporabo razvijalskih orodij v brskalniku Google Chrome. Že samo ta, na prvi pogled izredno majhna količina podatkov, predstavlja neverjetnih 62 TB uporabljene pasovne širine v celem mesecu. Če bi uspeli vsebino teh datotek zmanjšati samo za 1 KB, bi se končna mesečna poraba pasovne širine zmanjšala za občutnih 87 GB. Bolj obiskana kot je stran, bolj se majhne pridobitve v velikosti strani poznajo.

Med bolj znanimi in uporabljanimi metodami za minimizacijo je UglifyJS [22], ki pregleda podano datoteko JavaScript, iz nje odstrani vse presledke, odmike in komentarje ter imena funkcij in spremenljivk zamenja s krajšim imenom, po navadi z eno ali dvema črkama, kjer je to možno. Tako stisnjena vsebina je večinoma težko berljiva in skoraj neuporabna za nadaljnji razvoj in morebitne popravke, zato se ta proces izvaja ob koncu razvojnega cikla, pri čemer se hkrati poskrbi, da se z njim ne prepiše izvorna datoteka, ki jo lahko uporabimo za posodobitve v prihodnosti. Za primer učinkovitosti algoritma lahko vzamemo datoteko joke_of_day.js, ki jo spletni portal 24ur uporablja za prikaz vsakodnevne šale. Datoteka ni minimizirana in vsebuje 4368 znakov. Po izvedeni minimizaciji je vsebina velika le še 2357 znakov, kar je 53,9 % originalne velikosti oziroma približno 2 KB manj. Že samo s tem bi lahko 24ur.com prihranil približno 174 GB pasovne širine na mesec.

Datoteke JavaScript pa niso edine, ki se jih lahko minimizira. Tudi datoteke CSS in HTML lahko zmanjšamo in prihranimo nekaj dodatnih KB, tako da odstranimo vse nepotrebne odmike, presledke in komentarje. Tudi pri tem si lahko pomagamo z orodji, ki so na voljo na spletu, primer takega spletnega servisa je HTML Compressor [23], ki prepozna podano kodo in minimizira HTML, CSS in JavaScript.

5.2.4 Uporaba javnih knjižnic

Večina sodobnih spletnih strani uporablja več različnih knjižnic, s pomočjo katerih omogoča svojim uporabnikom različne funkcionalnosti in dosega želeno uporabniško izkušnjo. Nekatere izmed teh knjižnic so že tako razširjene, da so se pojavila omrežja CDN (*content delivery networks*), ki gostijo te knjižnice in nam omogočajo, da jih lahko vključimo v naše projekte. Primer takega servisa je tudi Googlov Google Hosted Libraries, ki ima na voljo nekaj popularnih knjižnic kot so jQuery, Prototype in Dojo. Vse, kar moramo narediti, je, da v naš dokument HTML dodamo naslednji košček kode in že imamo vse pripravljeno:

```
<script
src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
```

S tem pridobimo več pomembnih izboljšav. Prva je nedvomno ta, da nam ni potrebno servirati knjižnice z lastnega strežnika, zaradi česar prihranimo na porabljeni pasovni širini. Druga, ki je še posebej resnična v primeru Googlovega servisa, je, da se bo knjižnica zagotovo veliko hitreje prenesla iz Googlovih podatkovnih centrov, ki so na gosto posejani po celem svetu, kot pa z našega strežnika. Tretja in najpomembnejša izboljšava pa je izraba brskalnikovega predpomnilnika. Ker Googlove knjižnice uporabljajo tudi Googlove storitve in velik del spletnih strani, je verjetnost, da je uporabnik, ki je prišel na našo stran, v preteklosti že obiskal kakšno spletno stran, ki uporablja točno to knjižnico iz Googlovega servisa, zelo velika. Tak uporabnik ima to knjižnico že shranjeno v predpomnilniku svojega brskalnika, ki ob obisku naše spletne strani ugotovi, da je to knjižnico v preteklosti že prenesel na uporabnikov računalnik, in mu zato tega ni potrebno početi še enkrat. To pa pomeni, da brskalnik ne bo izvedel prenosa, in bo posledično naša spletna stran hitreje naložena.

6 Študija primera: mladi.net

Portal mladi.net je projekt društva Mavrični most mladih, neprofitne, nevladne organizacije, ki je osnovana na prostovoljstvu [24]. Cilj projekta je ponuditi mladim kristjanom neodvisen in odprt medij, kjer lahko neobremenjeno in anonimno sodelujejo v forumskih debatah, objavljajo članke in razmišljanja na aktualne dogodke in spremljajo dogajanje na katoliški mladinski sceni v Sloveniji. Skozi čas se je spletna stran razširila in v sodelovanju z drugimi organizacijami ponudila tudi strokovno anonimno spletno svetovanje v sodelovanju s strokovnjaki s številnih področij.

Sama spletna stran uporablja Joomlin sistem za upravljanje z vsebino, zadnjo oblikovno in vsebinsko prenovo pa je doživela leta 2007. Zaradi zastarelosti platforme in novih vsebinskih in tehničnih zahtev ekipe portala je prišlo do odločitve o prenovi in posodobitvi spletne strani, pri kateri kot član tehnične ekipe sodelujem tudi sam. V času pisanja tega diplomskega dela prenova še poteka in opisane rešitve še niso javno dostopne. V nadaljevanju se bo študija primera osredotočila samo na dele spletne strani, pri katerih so bili uporabljeni postopki odzivnega spletnega oblikovanja. Prenova sicer obsega tudi prenos in pretvorbo vsebine iz Joomlinega sistema za upravljanje z vsebino na Wordpress, kar pa za odzivno spletno oblikovanje ni pomembno.

6.1 Struktura spletne strani

Slika 8: Star videz spletne strani mladi.net

Začnimo z osnovno postavitvijo elementov na spletni strani. Star portal na Slika 8 ima glavni vsebinski blok s fiksno širino 990 pikslov, postavljen na sredino zaslona. Ta se deli na dva dela. Na levi strani imamo širši blok, v katerem se nahaja izpostavljena vsebina, kot so članki in manjši bloki s povezavami na vsebine iz drugih vsebinskih sklopov. Na desni strani pa imamo zaporedje manjših blokov: koledar; povezava na okrožnico; dnevne misli Afne, maskote spletne strani, in blok z oglasi. Pri širinah zaslonov 990 pikslov ali več je spletna stran povsem uporabna in preprosta za navigacijo. Na tablicah in mobilnih telefonih pa se

zaradi neodzivnosti dizajna stran skrči, da je v celoti prikazana na zaslonu. Posledica je skoraj nečitljiva vsebina in neintuitivna in časovno potratna navigacija, ki zahteva veliko približevanja in nepotrebnega premikanja po strani.

Slika 9: Koncept nove podobe portala mladi.net

Pri zasnovi nove podobe, ki je prikazana na Slika 9, smo uporabili Bootstrapovo knjižnico. Glavnemu vsebinskemu elementu smo pripeli razred `container-fluid`, njegovim naslednikom pa smo določili položaj v navidezni mreži s pomočjo razredov `row-fluid`, ki določajo vrstice, in s pomočjo razredov `span`, s pomočjo katerih določamo posamezne celice v vrstici. Posebnost razredov `span` je v tem, da s pomočjo dodanega števila na koncu imena

razreda določamo velikost celice, npr. `span8` ali `span4`. Vsaka vrsta je razdeljena na 12 stolpcev, skupni seštevek vseh elementov `span` mora biti torej enak 12, kar nam omogoči kombinacije razredov celic, kot so `span8` in `span4` ali `span4`, `span4`, `span4` ali `span2`, `span4`, `span6` itd. Kombinacij je veliko, zanje pa se odločimo na podlagi načrta naše spletne strani, ki predvideva koliko stolpcev bo naša spletna stran imela oziroma kako bodo posamezni vsebinski elementi razporejeni. Pravkar opisani Bootstrapovi razredi pa imajo še eno za nas zelo pomembno lastnost, na katero nakazuje že beseda *fluid* ki se nahaja v njihovem imenu. Ti razredi izvirajo iz Bootstrapove odzivne knjižnice in določajo razmerja med posameznimi elementi v vrstici v odstotkih in ne v fiksnih vrednostih. Zaradi tega so ti elementi odzivni in spreminjajo svoje dimenzije skladno s povečevanjem oziroma manjšanjem njihovega starša, elementa z razredom `container-fluid`. Temu staršu smo določili naslednje lastnosti:

```
.container-fluid {
   padding: 85px 0 0 0;
   width: 100%;
   max-width: 960px;
   margin: 0 auto;
}
```

Širina je vedno enaka širini brskalnikovega okna, vse do širine 960 pikslov, pri kateri se element neha širiti in ostaja enake velikosti. Določili smo tudi, da mora biti element vedno na sredini zaslona z enakomernim odmikom na levo in desno stran. To je vse, kar smo morali sami določiti, vse ostale ključne lastnosti, kot sta na primer dedovanje širine in ohranjanje razmerij pri naslednikih, pa so že določene z Bootstrapom.

Slika 10: Prikaz prilagojenega videza glede na velikost zaslona naprave

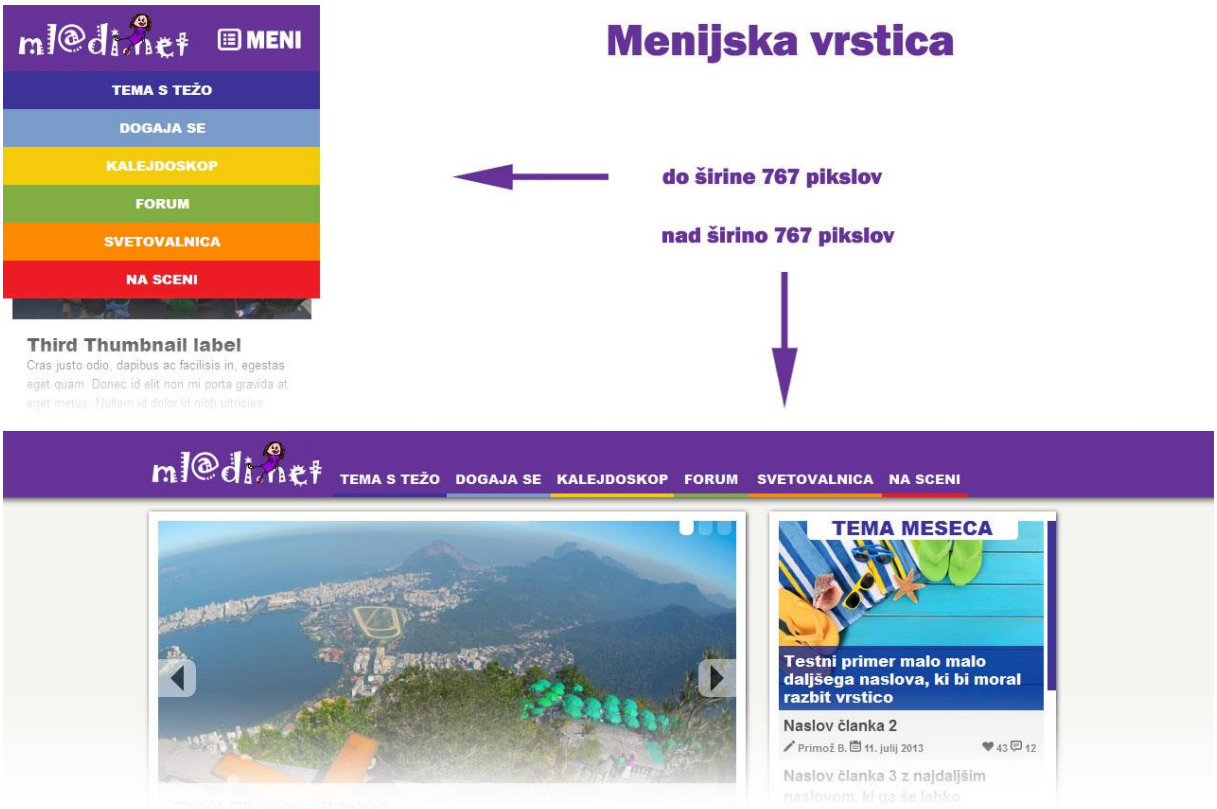
Stran je že odzivna, vendar s strukturo še nismo zadovoljni. V posamezni vrsti imamo po štiri vsebinske bloke. Taka struktura je na osebnih računalnikih in prenosnikih z večjim zaslonom odlična, ker uporabniku postreže z veliko informacijami, na manjših zaslonih pa zaradi relativnih razmerij med vsebinskimi bloki ti postajajo vedno manjši in zaradi svoje majhnosti vedno manj uporabni. Bootstrap ima za naprave z manjšim zaslonom že vgrajeno rešitev. Knjižnica sama poskrbi, da se pri zaslonih s širino, ki je manjša od 767 pikslov vsi elementi `span` razširijo na 100 % in zavzamejo celo vrstico. Vsebinski bloki, ki so bili na večjem zaslonu skupaj v isti vrsti, so zdaj prikazani drug pod drugim, kako je videti prilagojena postavitev vsebinskih blokov, pa je prikazano na Slika 10. Taka rešitev morda ni vedno najboljša, v našem primeru pa ustreza potrebam in željam pri razporeditvi vsebine.

6.2 Mobilnost na prvem mestu v praksi

Na manjših zaslonih naletimo zaradi manjšega prostora na še nekatere druge ovire, zaradi katerih moramo naš dizajn spremeniti. Ker smo spletno stran načrtovali po metodi progresivnega izboljševanja, imamo vse vsebinske sklope smiselno razdeljene, semantično obogatene in postavljene v svoje predvidene vsebinske bloke, kar je lepo razvidno iz strukture

spletne strani. Ti so z Bootstrapovo pomočjo umeščeni v tipografsko mrežo naše spletne strani, tej pa smo dodali še lastne prekrivne sloge, s katerimi smo vsebino obogatili. Za piko na i pa smo vsebinskemu bloku s poudarjenimi objavami dodali animirane prehode med novicami.

Oblikovno je dizajn zastavljen z mislijo na odzivnost in prilagodljivost. Vsi gradniki so preprosti in neodvisni drug od drugega, kar omogoča preprosto in hitro prestavljanje vsebinskih blokov v različne kompozicije. Taka zasnova je v skladu z mobilnostjo na prvem mestu, ki priporoča dizajn, ki se je sposoben dinamično nadgraditi in progresivno izboljševati skupaj z večanjem ločljivosti zaslona. Vsebinski bloki so pri najmanjših resolucijah postavljeni drug pod drugim, blok z oglasi, ki zavzema veliko prostora, pa je z uporabo Bootstrapovih razredov `hidden-tablet` in `hidden-phone` na teh resolucijah skrit in odstopa mesto drugim, vsebinsko pomembnejšim gradnikom.



Slika 11: Prilagajanje menijske vrstice glede na širino zaslona

Menijska vrstica iz Slika 11 je prav tako zasnovana po metodi mobilnosti na prvem mestu. Realizirana je kot seznam menijskih gumbov, postavljenih v stolpec. Pri zaslonih z dovolj

veliko ločljivostjo se ta seznam raztegne v vrstico, in tako prilagodi dodatnemu prostoru, ki ga ima na razpolago. Enako se zgodi z vsebinskimi bloki, ki se razporedijo iz stolpca v mrežo in se glede na vsebino, ki jo imajo, primerno povečajo. Blok s poudarjenimi objavami se tako razširi na dve tretjini širine prostora, namenjenega vsebinskemu delu, blok z rubriko Tema s težo pa prav tako postane malce večji od ostalih vsebinskih blokov.

V primerjavi s starim, statičnim dizajnom, je nova vizualna podoba spletne strani bistveno bolj prijazna uporabniku. Ker se velikost in razporeditev vsebine prilagaja ločljivosti zaslona, je ta vedno optimalno predstavljena in prostor smotrno izkoriščen. Zaradi odzivnega prilagajanja so vsebinski elementi smiselno razporejeni in dajejo vtis povezane celote, navigacija po taki strani je prav tako intuitivna in preprosta.

7 Sklepne ugotovitve

S pomočjo študije primera spletne strani mladi.net smo na praktičnem primeru prikazali uporabnost in prednosti odzivnega spletnega oblikovanja. Tehnologije in metode, ki ga sestavljajo, so še relativno nove, vendar v stalno spreminjajoči se podobi svetovnega spleta že pridobivajo na teži in prepoznavnosti. Odzivno spletno oblikovanje je predvsem z idejo o progresivnem izboljševanju odprlo novo poglavje na področju razvoja in dojemanja spletnih strani, ki pomeni večji preobrat v zgodovini svetovnega spleta. Miselnost, da so spletne strani prestižna dobrina, ki nam je ponujena v končni obliki, ki jo moramo sprejeti, kakršna koli že je, je preteklost. Spletne strani so postale osnovna pravica in nismo več uporabniki tisti, ki se moramo prilagajati tehničnim omejitvam spletnih strani. Ne, z vzponom odzivnega spletnega oblikovanja so to vlogo nase prevzele spletne strani same. Njihova naloga ni več samo podajanje vsebine, ampak prilagajanje celostne uporabniške izkušnje tehničnim omejitvam in zahtevam vsakega posameznega uporabnika spleta. In čeprav se bodo v naslednjih letih tehnologije, ki danes tvorijo odzivni splet, spremenile, zamenjale ali bodo morda celo ponovno definirale, kaj je to spletna izkušnja, bo usmeritev zagotovo ostala podobna, če ne celo enaka. Prihodnost spleta je usmerjenost k uporabniku in njegovi izkušnji in z odzivnim spletom se je ta miselnost šele začela v polnosti udejanjati.

Seznam slik in tabel

Literatura

[1] »HTML 5.1 Nightly«. http://www.w3.org/html/wg/drafts/html/master/ Pridobljeno 24. 8. 2013

[2] P. Gasston, »The Book of CSS3: A Developer's Guide to the Future of Web Design«, No Starch Press, San Francisco, 2011

[3] E. Marcotte, »Responsive Web Design«, A Book Apart, New York, 2011

[4] »Polyfill«. http://en.wikipedia.org/wiki/Polyfill Pridobljeno 13. 7. 2013

[5] »HTML5 Cross Browser Polyfills«. https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills Pridobljeno 22. 8. 2013

[6] »HTML5 Please«. http://html5please.com/#polyfill Pridobljeno 13. 7. 2013

[7] B. Frain, »Responsive Web Design with HTML 5 and CSS 3«, Packt Publishing, Birmingham, 2012

[8] »Top 10 Screen Resolutions from Mar 2009 to Sept 2013«. http://gs.statcounter.com/#resolution-ww-monthly-200903-201309 Pridobljeno 4. 9. 2013

[9] »Mobile vs. Desktop from Mar 2009 to Sept 2013«. http://gs.statcounter.com/#mobile_vs_desktop-ww-monthly-200812-201309 Pridobljeno 4. 9. 2013

[10] E. Meyer, »CSS Reset«. http://meyerweb.com/eric/tools/css/reset/ Pridobljeno 13. 7. 2013

[11] »Adaptive Images«. http://adaptive-images.com/details.htm Pridobljeno 13. 7. 2013

[12] »Fault-tolerant system«. http://en.wikipedia.org/wiki/Graceful_degradation Pridobljeno 13. 7. 2013

[13] A. Gustafson, »Understanding Progressive Enhancement«. http://alistapart.com/article/understandingprogressiveenhancement Pridobljeno 12. 7. 2013

[14] J. Grigsby, »Breaking Development Orlando 2013: Mobile First Responsive Design«. http://bdconf.com/2013/orlando/schedule#grigs Pridobljeno 22. 8. 2013

[15] »LESSCSS«. http://lesscss.org/ Pridobljeno 24. 8. 2013

[16] »Bootstrap«. http://getbootstrap.com/ Pridobljeno 24. 8. 2013

[17] »Modernizr«. http://modernizr.com/ Pridobljeno 24. 8. 2013

[18] »TinyPNG«. http://tinypng.org Pridobljeno 24. 8. 2013

[19] »CSS Image Sprites«. http://www.w3schools.com/css/css_image_sprites.asp Pridobljeno 24. 8. 2013

[20] »SpriteMe«. http://spriteme.org/ Pridobljeno 27. 7. 2013

[21] »MOSS«. http://www.moss-soz.si/si/rezultati_moss/obdobje/default.html Pridobljeno 28. 8. 2013

[22] »UglifyJS«.https://github.com/mishoo/UglifyJS Pridobljeno 12. 7. 2013

[23] »HTML Compressor«. http://htmlcompressor.com/compressor/ Pridobljeno 12. 7. 2013

[24] »mladi.net – kolofon«.http://www.mladi.net/content/view/1245/179/ Pridobljeno 9. 9. 2013

[25] L. Wroblewski, »Breaking Development: Mobile First Responsive Design«. http://www.lukew.com/ff/entry.asp?1707 Pridobljeno 28. 8. 2013

[26] S. Walter, »Content is Like Water«. Creative Commons Attribution-Share Alike 3.0 Unported. http://commons.wikimedia.org/wiki/File:Content_is_like_water.png Pridobljeno 24. 8. 2013

[27] Magictorch. »Polyfills«. http://magictorch.com/?p=1080 Pridobljeno 23. 8. 2013

[28] R. Dudler. »Relative Font Sizes for Web Designers«. http://frontify.com/blog/relative-font-sizes-for-web-designers/ Pridobljeno 24. 8. 2013

[29] I. Kobilica. »Kofetarica«. Olje na platnu. Javna last. http://sl.wikipedia.org/wiki/Slika:Ivana_Kobilca_-_Kofetarica.jpg Pridobljeno 24. 8. 2013

[30] E. Marcotte. (25. 5. 2010) »Responsive Web Design«. http://alistapart.com/article/responsive-web-design Pridobljeno 12. 7. 2013